Universidade do Estado Rio de Janeiro
Faculdade de Ciências Exatas e Engenharias
Campus Zona Oeste
Ciência da Computação
Programação Orientada a Objeto III

Sistema de Gestão Empresarial (ERP) com Java EE

Atividade Av1

Igor Rocha Rodrigues
Johan Carlos de Moura Miranda

Rio de Janeiro
2025

1. Introdução

1.1. Objetivo do Projeto:

O objetivo deste trabalho foi desenvolver uma plataforma de gestão empresarial (ERP) de média complexidade, 100% em Java, para um comércio varejista, utilizando as tecnologias JSF, Hibernate (JPA) e EJB. O sistema foi projetado para ser modular, seguro e preparado para um ambiente de produção simulado com Docker.

1.2. Escopo das Funcionalidades:

A aplicação contempla o gerenciamento completo de usuários, perfis de acesso, produtos, categorias, fornecedores, compras e vendas, além de funcionalidades de segurança avançada, relatórios e um dashboard interativo.

2. Tecnologias e Ferramentas Utilizadas

Linguagem: Java 11

Frontend: Jakarta Server Faces (JSF) 2.3 com a biblioteca de componentes PrimeFaces 11.0.

Backend (Lógica de Negócio): Enterprise JavaBeans (EJB) 3.2.

Persistência de Dados: Java Persistence API (JPA) com a implementação do Hibernate 5.3.

Banco de Dados: MySQL 8.0.

Servidor de Aplicação: WildFly 26.1.3.Final.

Ambiente e Deploy: Docker e Docker Compose para orquestração dos contêineres da aplicação e do banco de dados.

Build e Dependências: Apache Maven.

Bibliotecas Adicionais: OmniFaces 3.13 para utilitários JSF.

3. Arquitetura da Solução

3.1. Visão Geral:
O sistema foi construído em uma arquitetura de três camadas (Three-Tier):

Camada de Apresentação (Frontend): Responsável pela interface com o usuário, construída com páginas XHTML (Facelets) e componentes PrimeFaces. Os Managed Beans (Controllers) fazem a ponte entre a visão e a lógica de negócio.

Camada de Negócio (Backend): A lógica de negócio, como validações e orquestração de operações, é encapsulada em EJBs @Stateless. Isso garante transacionalidade e escalabilidade.

Camada de Persistência: O acesso ao banco de dados é abstraído pelo JPA. As entidades são classes Java anotadas (@Entity), e o EntityManager é utilizado dentro dos EJBs de serviço para realizar as operações de CRUD.

3.2. Containerização com Docker:
A aplicação é 100% conteinerizada, utilizando um Dockerfile customizado para construir a imagem da aplicação (com WildFly e JDK) e um arquivo docker-compose.yml para orquestrar a subida dos serviços da aplicação (wildfly-app) e do banco de dados (mysql-db), garantindo um ambiente de desenvolvimento e produção consistente.

4. Modelo de Dados (Entidades JPA)

User: Representa os usuários do sistema (administradores, funcionários e clientes).

Role: Representa os perfis de acesso (ADMINISTRADOR, FUNCIONARIO, CLIENTE).

Produto: Representa os produtos à venda, com estoque, preço e relacionamentos.

Categoria: Agrupa os produtos (ex: Periféricos, Monitores).

Fornecedor: Representa as empresas que fornecem os produtos.

Venda e ItemVenda: Registram as transações de venda e os produtos vendidos em cada uma.

Compra e ItemCompra: Registram as transações de compra de fornecedores.

AuditLog: Armazena os logs de atividades importantes dos usuários.

5. Funcionalidades Mínimas Implementadas

Esta seção detalha como cada um dos requisitos mínimos solicitados no escopo do projeto foi implementado, conectando a teoria à prática do código desenvolvido.

5.1. Controle de Acesso e Segurança Avançada

Autenticação e Autorização (RBAC): Implementamos um robusto sistema de controle de acesso.

Autenticação: O LoginController.java, em conjunto com o EJB AuthServiceImpl.java, valida as credenciais do usuário contra o banco de dados. Um User logado é mantido em escopo de sessão.

Autorização: A classe AuthorizationFilter.java intercepta todas as requisições. Ela verifica se o usuário está logado e qual o seu Role (ADMINISTRADOR, FUNCIONARIO ou CLIENTE) para permitir ou negar o acesso a URLs específicas, redirecionando para a página acesso-negado.xhtml quando necessário.

Recuperação de Senha: O UserController.java e o UserServiceImpl.java contêm a lógica para gerar um token de redefinição de senha com data de expiração. O envio do token é simulado no console do servidor através do EmailService.java, conforme permitido pelo escopo do projeto. As telas forgotPassword.xhtml e resetPassword.xhtml compõem a interface dessa funcionalidade.

Auditoria e Logs de Acesso: Criamos a entidade AuditLog.java e o AuditLogService para registrar ações críticas, como tentativas de login (bem-sucedidas ou falhas) e o encerramento de sessão, garantindo a rastreabilidade das ações no sistema.

5.2. Gestão de Usuários e Perfis
Cadastro e Gerenciamento de Usuários:

Administradores: A tela usuarios.xhtml, controlada pelo UserController.java, permite que administradores realizem o CRUD (Criar, Ler, Atualizar, Deletar) completo de todos os usuários do sistema.

Clientes: Implementamos uma tela pública de auto-registro (registro.xhtml), controlada pelo RegistroController.java, onde clientes podem criar suas próprias contas, que são automaticamente associadas ao perfil "CLIENTE".

Gerenciamento de Perfis: Na tela de edição de usuários, um administrador pode associar múltiplos perfis (ADMINISTRADOR, FUNCIONARIO, CLIENTE) a um usuário através de um componente <p:selectManyCheckbox>, garantindo a flexibilidade do sistema de permissões.

5.3. Gestão de Produtos e Estoque
CRUD de Produtos e Categorias:

O sistema possui telas dedicadas para o gerenciamento completo de Produtos (produtos.xhtml) e Categorias (categorias.xhtml).

A entidade Produto.java possui relacionamentos @ManyToOne com Categoria.java e Fornecedor.java, permitindo uma organização robusta dos itens.

Controle de Estoque: A lógica de atualização de estoque é transacional e implementada nos EJBs. O estoque é incrementado no CompraServiceImpl.java ao registrar uma compra e decrementado no VendaServiceImpl.java ao finalizar uma venda. A tela de produtos também exibe um alerta visual (cor vermelha) para itens com baixo estoque.

5.4. Gestão de Fornecedores e Compras
CRUD de Fornecedores: A tela fornecedores.xhtml permite o cadastro completo de fornecedores, gerenciado pelo FornecedorController.java e FornecedorService.

Registro de Compras: A tela compras.xhtml, controlada pelo CompraController.java, simula o processo de compra de um fornecedor. O usuário seleciona um fornecedor e adiciona produtos a um carrinho de compra, que ao ser finalizado, atualiza o estoque dos produtos correspondentes.

5.5. Gestão de Vendas e Pagamentos
Checkout de Vendas (Frente de Caixa): A página vendas.xhtml funciona como um terminal de vendas.

Carrinho: O VendaController.java gerencia um carrinho de compras em memória.

Associação de Cliente: Funcionários e administradores podem associar a venda a um cliente cadastrado através de um menu de seleção. Se o usuário logado for um cliente, ele é associado automaticamente à sua própria compra.

Formas de Pagamento: A tela apresenta um menu de seleção (<p:selectOneRadio>) para que o vendedor escolha a forma de pagamento (Cartão de Crédito, PIX, Boleto, Dinheiro).

Emissão de Recibo: Ao finalizar uma venda, o usuário é redirecionado para a página recibo.xhtml. Esta página apresenta todos os dados da transação (cliente, itens, total, forma de pagamento) em um formato claro para impressão e oferece um botão para baixar um arquivo .txt do recibo, gerado dinamicamente pelo ReciboController.java.

Dashboard Interativo: A página principal de vendas (vendas.xhtml) exibe um gráfico de barras do PrimeFaces que mostra o faturamento total por dia, fornecendo uma visão gerencial rápida do desempenho do negócio. Este gráfico é renderizado apenas para funcionários e administradores.

6. Contribuições Individuais

O desenvolvimento do projeto foi um esforço colaborativo e integrado, com ambos os membros do grupo participando ativamente de todas as fases, desde a concepção e arquitetura até a implementação e depuração. As tarefas foram realizadas em conjunto, e o código foi revisado constantemente, garantindo que ambos os integrantes tivessem um conhecimento profundo de todas as partes do sistema.

7. Instruções para Execução do Projeto

Pré-requisitos: Docker e Docker Compose instalados, e Maven para compilar o projeto.

Compilação: Na pasta raiz do projeto, execute o comando mvn clean install para gerar o arquivo trabalhoav1.war.

Execução: Na mesma pasta, execute o comando docker-compose up --build. Os contêineres do banco de dados e da aplicação serão iniciados.

Acesso: A aplicação estará disponível no endereço http://localhost:8080/trabalhoav1/.

Credenciais Padrão:

Admin: admin / admin

Funcionário: func / func

Cliente: cliente / cliente